好的，很高兴能为您设计一款基于 JavaFX 的高校教材购销系统框架。这个框架将遵循现代软件工程的分层思想，确保代码的模块化、可维护性和可扩展性。

### 高校教材购销系统框架设计

本系统旨在为高校学生、教师和管理员提供一个方便、高效的教材购买、选用和管理平台。

#### 1. 系统总体架构 (Overall Architecture)

我们将采用经典的三层架构模型，将系统清晰地划分为 **表现层 (UI)**、**业务逻辑层 (Service)** 和 **数据访问层 (DAO)**。

+-------------------------------------------------+  
| 表现层 (UI Layer) |  
| - JavaFX, FXML, Controllers |  
| - 负责用户界面展示和交互 |  
+----------------------+--------------------------+  
 | (调用)  
+----------------------v--------------------------+  
| 业务逻辑层 (Service Layer) |  
| - 处理核心业务逻辑 (如订单、库存) |  
| - 事务管理 |  
+----------------------+--------------------------+  
 | (调用)  
+----------------------v--------------------------+  
| 数据访问层 (DAO Layer) |  
| - 封装对数据库的增删改查 (CRUD) 操作 |  
+----------------------+--------------------------+  
 | (连接)  
+----------------------v--------------------------+  
| 数据库 (Database) |  
| - MySQL 5.x |  
+-------------------------------------------------+  
| 实体模型层 (Model/Entity) |  
| - 在各层之间传递的数据对象 (POJO) |  
+-------------------------------------------------+

* **表现层 (UI)**: 使用 JavaFX 和 FXML 构建。FXML 文件负责定义界面布局，Controller 类负责处理用户输入和事件，并调用业务逻辑层。这种方式能让界面设计与程序逻辑分离。
* **业务逻辑层 (Service)**: 系统的核心。它处理所有复杂的业务规则，例如：创建一个新订单需要同时扣减库存和记录订单信息。它会调用一个或多个 DAO 接口来完成任务。
* **数据访问层 (DAO - Data Access Object)**: 负责与 MySQL 数据库直接交互。每一张数据库表通常对应一个 DAO 接口及其实现类。它将底层的 JDBC 操作封装起来，向上层提供简洁的数据操作方法。
* **实体模型层 (Model)**: 普通的 Java 对象 (POJO)，用于在各层之间传递数据。每个 Model 类通常对应数据库中的一张表。

#### 2. 数据库设计 (MySQL Database Design)

以下是核心表的初步设计，您可以根据具体需求进行扩展。

1. **用户表 (t\_user)**: 存储所有用户（学生、教师、管理员）。
   * id (INT, 主键, 自增)
   * username (VARCHAR, 唯一, 登录名)
   * password (VARCHAR, 存储加密后的密码)
   * role (ENUM('STUDENT', 'TEACHER', 'ADMIN'), 角色)
   * name (VARCHAR, 真实姓名)
   * student\_id (VARCHAR, 学号，可选)
   * create\_time (TIMESTAMP)
2. **教材表 (t\_book)**: 存储教材信息。
   * id (INT, 主键, 自增)
   * isbn (VARCHAR, 国际标准书号, 唯一)
   * title (VARCHAR, 书名)
   * author (VARCHAR, 作者)
   * publisher (VARCHAR, 出版社)
   * price (DECIMAL, 价格)
   * stock (INT, 库存数量)
   * description (TEXT, 描述)
3. **院系表 (t\_department)**: 存储院系信息。
   * id (INT, 主键, 自增)
   * name (VARCHAR, 院系名称)
4. **课程表 (t\_course)**: 存储课程信息。
   * id (INT, 主键, 自增)
   * course\_code (VARCHAR, 课程代码)
   * course\_name (VARCHAR, 课程名称)
   * department\_id (INT, 外键, 关联 t\_department.id)
   * teacher\_id (INT, 外键, 关联 t\_user.id)
5. **课程-教材关联表 (t\_course\_book)**: 描述课程需要哪些教材（多对多关系）。
   * id (INT, 主键, 自增)
   * course\_id (INT, 外键, 关联 t\_course.id)
   * book\_id (INT, 外键, 关联 t\_book.id)
6. **订单表 (t\_order)**: 记录学生的购买订单。
   * id (INT, 主键, 自增)
   * order\_number (VARCHAR, 订单号, 唯一)
   * student\_id (INT, 外键, 关联 t\_user.id)
   * total\_price (DECIMAL, 订单总价)
   * status (ENUM('PENDING', 'PAID', 'SHIPPED', 'CANCELLED'), 订单状态)
   * create\_time (TIMESTAMP)
7. **订单详情表 (t\_order\_detail)**: 记录一个订单具体包含哪些书。
   * id (INT, 主键, 自增)
   * order\_id (INT, 外键, 关联 t\_order.id)
   * book\_id (INT, 外键, 关联 t\_book.id)
   * quantity (INT, 购买数量)
   * unit\_price (DECIMAL, 购买时的单价)

#### 3. 核心功能模块设计

根据用户角色，系统可以划分为以下主要功能模块：

**A. 通用模块**

* **登录/注册模块**:
  + UI: LoginView.fxml, RegisterView.fxml
  + 逻辑: 验证用户身份，根据角色跳转到不同主界面。

**B. 学生端模块**

* **教材浏览与搜索**: 按课程、书名、作者等搜索教材。
* **购物车管理**: 添加教材到购物车、修改数量、删除。
* **订单中心**: 生成订单、在线支付（可模拟）、查看历史订单。
* **个人信息管理**: 修改个人资料。

**C. 教师端模块**

* **我的课程管理**: 查看自己教授的课程列表。
* **教材选用**: 为自己的课程指定/推荐教材（即操作 t\_course\_book 表）。

**D. 管理员端模块**

* **用户管理**: 添加、删除、修改用户信息，重置密码。
* **教材管理**: 入库新教材、修改教材信息、调整库存。
* **订单管理**: 查看所有订单、处理发货、更新订单状态。
* **院系与课程管理**: 管理院系和课程信息。
* **数据统计**:（可选）生成销售报表等。

#### 4. 技术实现细节与建议

* **环境**:
  + JDK 1.8
  + MySQL 5.5.40+
  + IDE: IntelliJ IDEA 或 Eclipse (需安装 e(fx)clipse 插件)
  + 构建工具: Maven 或 Gradle (推荐 Maven，便于管理依赖)
* **JavaFX (UI层)**:
  + 强烈建议使用 **FXML** 来设计界面。可以使用 **Scene Builder** 工具通过拖拽方式设计 FXML，提高效率。
  + 为每个 FXML 文件创建一个对应的 Controller 类。
  + 主程序通过加载 FXML 文件来构建场景 (Scene) 和舞台 (Stage)。
* **数据库连接 (DAO层)**:
  + 使用 **JDBC** 直接连接数据库。你需要下载并引入 mysql-connector-java 的依赖。
  + 为了提高性能和代码健壮性，可以引入一个数据库连接池，如 **HikariCP** 或 **C3P0**。
* 项目结构 (Maven):  
  一个清晰的项目结构至关重要。  
  src  
  ├── main  
  │ ├── java  
  │ │ └── com  
  │ │ └── yourcompany  
  │ │ └── bookstore  
  │ │ ├── MainApplication.java // JavaFX 启动类  
  │ │ ├── controller // UI 控制器  
  │ │ │ ├── LoginController.java  
  │ │ │ └── ...  
  │ │ ├── dao // 数据访问对象接口  
  │ │ │ └── impl // DAO 实现类  
  │ │ ├── model // 实体模型  
  │ │ │ ├── User.java  
  │ │ │ └── Book.java  
  │ │ ├── service // 业务逻辑服务接口  
  │ │ │ └── impl // Service 实现类  
  │ │ └── util // 工具类 (如DBUtil)  
  │ └── resources  
  │ └── com  
  │ └── yourcompany  
  │ └── bookstore  
  │ ├── view // FXML 文件  
  │ │ ├── LoginView.fxml  
  │ │ └── MainView.fxml  
  │ └── images // 图片资源  
  pom.xml // Maven 配置文件
* UI详细设计：

好的，我们来具体设计一下这款系统在 UI 层面（JavaFX）的呈现方式和交互流程。

UI 设计的目标是：**简洁、直观、符合用户习惯、角色分明**。

我们将采用单窗口、多视图的设计模式。即一个主窗口 (Stage)，根据用户的操作在主窗口中心区域切换不同的视图 (View/Scene)。

**1. 整体布局与风格**

**主窗口布局 (MainView.fxml)**

建议使用 BorderPane 作为主窗口的根布局，因为它非常适合典型的应用程序结构。

* **TOP (顶部区域)**: HBox 或 ToolBar。
  + 左侧显示系统Logo和名称，如 “高校教材购销系统”。
  + 右侧显示当前登录用户的姓名和角色（如 “欢迎您，张三 (学生)”），以及一个 “退出登录” 按钮。
* **LEFT (左侧区域)**: VBox 或 ListView。
  + 这是主导航栏。根据登录用户的角色（管理员、教师、学生），显示不同的功能按钮。
* **CENTER (中心区域)**: StackPane 或 AnchorPane。
  + 这是核心内容显示区。点击左侧导航栏的按钮后，对应的功能界面 (FXML) 会被加载到这个区域。
* **BOTTOM (底部区域)**: HBox。
  + 用作状态栏，可以显示一些提示信息、当前时间或版权信息。

**UI 风格**

* **CSS 样式**: 强烈建议为你的应用创建一个全局的 CSS 文件 (style.css)。
  + 定义统一的颜色方案（如蓝色主题）、字体大小、按钮样式（如鼠标悬停效果）、表格样式等。
  + 这样可以轻松地改变整个应用的外观，而无需修改 FXML 文件。
* **图标**: 在按钮和导航项上使用清晰的图标（如使用 FontAwesomeFX 库），可以极大地提升用户体验。

**2. 关键界面设计 (Text-based Wireframe)**

**A. 登录界面 (LoginView.fxml)**

这是应用的入口。

* **布局**: GridPane 置于窗口中央，周围留有边距。
* **组件**:
  + ImageView: (可选) 显示校徽或系统 Logo。
  + Label: "用户登录"。
  + TextField: 用户名输入框 (提示文字: “请输入学号/工号”)。
  + PasswordField: 密码输入框。
  + ComboBox<String>: 角色选择 (包含 “学生”, “教师”, “管理员”)。
  + CheckBox: "记住我" (可选)。
  + Button: "登 录" (设置为默认按钮，按回车即可触发)。
  + Hyperlink: "注册新用户" (点击后弹出注册对话框)。
  + Label: 用于显示错误信息，如 "用户名或密码错误！"。

**B. 学生端主界面**

当学生登录后，主窗口的左侧导航栏和中心内容区会显示为学生定制的视图。

* **左侧导航栏 (VBox)**:
  + Button: "教材浏览与购买" (默认选中)
  + Button: "我的购物车"
  + Button: "我的订单"
  + Button: "个人中心"
* **中心内容区 - 教材浏览 (BookBrowseView.fxml)**:
  + **顶部**: 一个 HBox 作为搜索/过滤栏。
    - ComboBox: 按院系筛选。
    - ComboBox: 按课程筛选。
    - TextField: 输入框，用于按书名、ISBN、作者搜索。
    - Button: "搜索"。
  + **中部**: TableView 用于显示教材列表。
    - **列**: ISBN, 书名, 作者, 出版社, 价格, 库存。
    - **操作列**: 每一行末尾都有一列，包含一个 "加入购物车" Button。如果库存为0，此按钮应被禁用。
  + **底部**: Pagination 控件，用于在教材过多时进行分页显示。
* **中心内容区 - 购物车 (CartView.fxml)**:
  + TableView 显示已加入购物车的商品。
    - **列**: 书名, 单价, 数量 (数量列可以放一个 Spinner 或 TextField 允许用户修改), 小计。
    - **操作列**: "删除" Button。
  + **底部**: HBox 显示信息。
    - Label: "商品总数: X"。
    - Label: "合计: ￥YYY.YY"。
    - Button: "清空购物车"。
    - Button: "去结算" (高亮显示)。点击后弹出确认对话框，然后生成订单。

**C. 管理员端主界面**

管理员登录后，导航栏功能强大，中心内容区主要是数据管理表格。

* **左侧导航栏 (Accordion 或 TitledPane 分组)**:
  + **教材管理 (分组)**
    - Button: "教材列表"
    - Button: "教材入库"
  + **订单管理 (分组)**
    - Button: "订单查询"
  + **用户管理 (分组)**
    - Button: "用户列表"
  + **系统设置 (分组)**
    - Button: "院系课程管理"
* **中心内容区 - 教材管理 (AdminBookManageView.fxml)**:
  + **顶部**:
    - Button: "新增教材" (点击后弹出新的窗口或 Dialog 用于填写信息)。
    - 搜索框等筛选条件。
  + **中部**: TableView 显示所有教材。
    - **列**: ID, ISBN, 书名, 作者, 价格, 库存。
    - **操作列**: 包含两个按钮 "编辑" 和 "删除"。
      * 点击 "编辑"，弹出与 "新增教材" 类似的对话框，但已填充好当前行的数据。
      * 点击 "删除"，弹出 Alert.AlertType.CONFIRMATION 对话框，询问 “您确定要删除这本教材吗？”。
* **新增/编辑教材对话框 (EditBookDialog.fxml)**:
  + **布局**: GridPane，左边是 Label，右边是输入控件。
  + **组件**:
    - TextField: 书名
    - TextField: 作者
    - TextField: 出版社
    - TextField: ISBN
    - Spinner<Double>: 价格
    - Spinner<Integer>: 库存
    - TextArea: 描述
    - Button: "保存"
    - Button: "取消"

**3. 交互与用户体验 (UX)**

* **动态加载**: 点击左侧导航按钮时，只刷新 BorderPane 的中心区域，而不是整个窗口。这可以通过加载不同的 FXML 文件并将其设置为中心节点来实现。
* **反馈及时**:
  + 执行耗时操作（如查询大量数据）时，显示一个进度指示器 (ProgressIndicator)。
  + 操作成功后，使用 Alert 对话框给出提示，例如 "订单提交成功！" (Alert.AlertType.INFORMATION)。
  + 输入验证失败或操作出错时，也使用 Alert 给出明确的错误信息 (Alert.AlertType.ERROR)。
* **数据验证**: 在用户提交表单（如新增教材、注册）时，对输入内容进行前端验证。例如，价格和库存必须是数字，关键信息不能为空。不符合要求的输入框可以用红色边框高亮提示。
* **一致性**: 整个应用的对话框样式、按钮位置（如“确定”总是在右边）应保持一致。

通过以上设计，你可以构建出一个功能清晰、界面友好、体验流畅的高校教材购销系统。在实现时，可以先从最核心的功能（如管理员的教材管理和学生的教材浏览）开始，逐步完善其他模块。

#### 5. 开发步骤建议

1. **环境搭建**: 安装 JDK 1.8, MySQL, Maven, 和 IDE。
2. **数据库先行**: 在 MySQL 中创建数据库和上述设计的表结构。
3. **创建项目**: 使用 Maven 创建一个新的 JavaFX 项目。在 pom.xml 中添加 mysql-connector-java 和 javafx 相关依赖。
4. **编码 (自底向上)**:
   * 先编写 **Model** 层实体类。
   * 再编写 **DAO** 层，并编写单元测试来验证数据库操作的正确性。
   * 然后开发 **Service** 层，调用 DAO 完成业务逻辑。
   * 最后开发 **UI** 层，使用 Scene Builder 设计 FXML，并编写 Controller 调用 Service。
5. **集成与测试**: 将所有模块连接起来，进行全面的功能测试。
6. **打包与部署**: 使用 Maven 的打包插件 (如 maven-shade-plugin 或 javafx-maven-plugin) 将应用打包成一个可执行的 JAR 文件。

这个框架为您提供了一个坚实的起点。祝您项目开发顺利！